Don't Like RDF Reification? Creating Meta Triples Describing Triples Using Singleton Property

Vinh Nguyen[1], Olivier Bodenreider[2], Amit Sheth[1]

[1] Kno.e.sis Center, Wright State University, Ohio, USA
[2] National Library of Medicine, Bethesda, Maryland, USA

Abstract. Meta triples provide additional information about individual triples, such as the source, the occuring time or place, or the certainty. Integrating such meta triples into semantic knowledge bases would enable the querying and reasoning mechanisms to be aware of provenance, time, location, or certainty of triples. However, an efficient RDF representation for such meta knowledge of triples remains challenging. The existing reification approach allows such meta knowledge of RDF triples to be expressed using RDF by two steps. The first step is representing the triple by a Statement instance which has subject, predicate, and object indicated separately in three different triples. The second step is creating assertions about that instance as if it is a statement. While reification is simple and intuitive, this approach does not have formal semantics and is not commonly used in practice as described in the RDF Primer.
In this paper, we propose a novel approach called *Singleton Property* for representing meta triples and provide a formal semantics for it. We explain how this singleton property approach fits well with the existing syntax and formal semantics of RDF, and the syntax of SPARQL query language. We also demonstrate the use of singleton property in the representation and querying of meta knowledge in two examples of Semantic Web knowledge bases: YAGO2 and BKR. This approach, which is also simple and intuitive, can be easily adopted for representing and querying meta triples in other knowledge bases.

1 Introduction

Representing and querying meta knowledge for triples including provenance, trust, certainty, time, and location have been an emerging demand in creating and sharing Semantic Web knowledge bases [8, 15–17, 19]. Here we use the example from YAGO2[8] for demonstrating the requirements of meta knowledge for triples and motivating our approach.

1.1 Motivating scenario

Resource Description Framework (RDF) [5, 12] has been well adopted for creating and sharing various knowledge bases. Knowledge bases that provide a comprehensive collection of facts (e.g., YAGO [20]) have been widely used by various applications. These facts are usually in the form of triples, or subject-predicate-object such as:

```
BobDylan     isMarriedTo              SaraLownds
BarackObama  holdsPoliticalPosition   PresidentOfTheUnitedStates.
```

While these facts are useful for finding spouses or political positions of a person, they do not provide sufficient information for answering many types of challenging questions involving meta knowledge. Such list of query types and their examples are listed in Table 1. Additional information about the triples must be provided in order to address those queries. Recent knowledge bases such as YAGO2 [8] provide such temporal and spatial information. For example, the information about the sources and dates of the fact `BobDylan isMarriedTo SaraLownds` would help find the answers for question P1 (where is the fact from?) and P2 (when was it created?) in Table 1.

Table 1. Sample queries for different types of meta knowledge, each query example is assigned an identifier (P, T, S, and C) for reference

Query type	**Examples**
Provenance query	P1. Where is this fact from?
	P2. When was it created?
	P3. Who created this fact?
Temporal query	T1. When did this fact occur?
	T2. What is the time span of this fact?
	T3. Which events happened in the same year?
Spatial query	S1. What is the location associated with this fact?
	S2. Which events happened at the same place?
Certainty query	C1. What is the author confidence of this fact?

We assume that this fact could be extracted from the wiki pages of Bob_Dylan on 2009-06-07 and Sara_Dylan on 2009-08-08. Using the reification approach, the whole reified statements and their assertions about sources and extraction dates are illustrated in Table 2. The fact is represented as an instance of class Statement with three different properties for its subject, property and object. Since we need to represent two occurences of the same statement in two different documents, we create two resources: `stmt#1` and `stmt#2` because if we create only one `stmt#1` for both occurrences, the association of each occurrence with its source and date of extraction together is not distinguishable. The meta information about the fact is represented by `hasSource` and `extractedOn` properties.

The lack of formal semantics connecting a statement and the resource describing it is one of the main drawbacks of using reification for describing triples. Since the resource `stmt#1` describing a statement is not associated with that statement, assertions created for this resource are not the same as assertions created for the original statement as explained in [12, 5]. Moreover, it is obvious

Table 2. Reified statements and their meta knowledge assertions for the same fact `BobDylan isMarriedTo SaraLownds` occuring in two documents

Subject	Predicate	Object	Subject	Predicate	Object
stmt#1	type	Statement	stmt#2	type	Statement
stmt#1	hasSubject	BobDylan	stmt#2	hasSubject	BobDylan
stmt#1	hasProperty	isMarriedTo	stmt#2	hasProperty	isMarriedTo
stmt#1	hasObject	SaraLownds	stmt#2	hasObject	SaraLownds
stmt#1	*hasSource*	wk:Bob_Dylan	stmt#2	*hasSource*	wk:Sara_Dylan
stmt#1	*extractedOn*	2009-06-07	stmt#2	*extractedOn*	2009-08-08

that the reification approach requires four additional triples for representing one statement per document as a resource. This would increase the size of the data sets by at least four times, which is not a scalable approach. It would also make query patterns lengthy for finding when the couple was married or divorced.

1.2 Our approach

In this paper, we address the problem of representing and querying meta knowledge of triples by looking at it from a different perspective. Our motivation rises from the question as to whether or not a more efficient mechanism for describing a statement using RDF exists. A good design should provide a formal semantics, use existing syntax and be compatible with existing Semantic Web languages, tools, and methods. Firstly, the proposed formal semantics should be compatible with the existing model-theoretic semantics to avoid conflicts in the RDF/RDFS interpretation. Secondly, using existing syntax would ensure the compatibility of meta triples and existing triple datasets. Finally, it would overcome the need to develop new or revise available tools and methods for making them work with new meta triples.

This paper proposes a novel approach called *Singleton Property* for representing meta statements using RDF with regard to the three requirements above. The intuition of our approach is based on the assumption that the nature of every relationship is universally unique. The uniqueness of the relationship can be a key for any statement using the new type of property called *singleton property*. A singleton property is a property instance representing one specific relationship between two particular entities under one specific context. Singleton properties can be viewed as instances of generic properties whose extensions contain a set of entity pairs. Similar to the way we assign label for generic property, we assign a unique label for each singleton property.

For example, for the same statement `BobDylan isMarriedTo SaraLownds`, we can create two singleton property instances describing the occurrences of this statement in two documents as illustrated in Table 3. For each document (or context of the fact), we create one separate singleton property instance representing that fact (in T_1, T_5). Particularly we create `isMarriedTo#1` and `isMarriedTo#2` for the relationships extracted from Wiki pages of Bob Dylan (`wk:Bob_Dylan`)

Table 3. Singleton properties and their meta knowledge assertions for the same fact `BobDylan isMarriedTo SaraLownds` occuring in two documents

Triple N_o	**Subject**	**Predicate**	**Object**
T_1	BobDylan	isMarriedTo#1	SaraLownds
T_2	isMarriedTo#1	rdfs:subPropertyOf	isMarriedTo
T_3	isMarriedTo#1	*hasSource*	wk:Bob_Dylan
T_4	isMarriedTo#1	*extractedOn*	2009-06-07
T_5	BobDylan	isMarriedTo#2	SaraLownds
T_6	isMarriedTo#2	rdfs:subPropertyOf	isMarriedTo
T_7	isMarriedTo#2	*hasSource*	wk:Sara_Dylan
T_8	isMarriedTo#2	*extractedOn*	2009-08-08

and Sara Dylan (`wk:Sara_Dylan`), respectively. Meta knowledge about the fact from one document can be added as assertions for the singleton property from that document respectively (in T_3, T_4, T_7, and T_8). These two singleton properties as sub properties of the same generic property `isMarriedTo` (in T_2, T_6) allows us to infer the original statement. Particularly, one can easily infer the original statement `BobDylan isMarriedTo SaraLownds` from statements T_1 and T_2, or T_5 and T_6, using RDFS entailment rule of subPropertyOf [5]. One drawback of instantiating singleton property via `rdfs:subPropertyOf` is that it adds too many leaf nodes into the property hierarchy. This can be avoided by instantiating the singleton property via `rdf:type` (in T_2 and T_6). Using this `rdf:type` instantiation method, generic properties become binary classes and singleton properties become instances of these binary classes. We will provide a more detailed explanation on how we come up with the set of triples listed in Table 3 in the next section.

The remaining sections are organized as follows. Section 2 explains the approach in details and justifies our design choices for the singleton property. Section 3 provides a formal semantics for the singleton property. The querying mechanism is described in Section 4. We also discuss issues related to how this singleton property could be adopted by existing Semantic Web technologies and standards by providing two use cases in the BKR[15] and YAGO2.

2 Singleton Property

In this section, we will explain our intuition for the proposed approach and justify our design choices for the singleton property.

Back to the motivating example we used, the reification process represents a triple as a resource, which is an instance of the Statement class [5]. Reifying a statement requires two steps. The first step is to find a resource that uniquely identifies a statement. The second is to create assertions for that statement via that resource. Here we explain our rationale accounting for the novel perspective that leads to our approach.

2.1 Singleton property as unique key for statement within a context

The first step involves finding which resource among the three elements of a triple could fundamentally distinguish statements.

In the Semantic Web, everyone can create any statement. It is possible that the same statements may be created in different datasets by different organizations. Therefore, we need to find a resource that can distinguish any two statements. Given that the statements may be same, they may be associated with different contextual information when they are created. The information capturing the context when a statement is created could be helpful for identifying statements. Such contextual information of a statement could be described by various dimensions of meta knowledge, including the source recording that statement, the time or place that statement occurs, the certainty of the author about that statement, and etc. We can conclude that a statement within a context is unique. Now the next question is, what can represent that uniqueness of a statement within a context? If the same statements are associated with different contexts, are they the same in nature? What remains same? What becomes different?

From a philosophical point of view, we believe that the existence of two entities in the subject and the object of one statement is independent from the contexts creating that statement. Particularly, they already exist before the statement is created. For example, the existence of Bob Dylan and Sara Lownds does not depend on their marriage, and obviously they also exist before they marry each other. While creating a new statement, what we actually do is connect two existing entities and establish a new relationship between them. Therefore, the contextual information while establishing a new relationship can play the role of a key for any statement. We can manifest that key by creating a new property instance representing the newly established relationship associated with a context and enforcing it to be unique. We call it singleton property. The *singleton* concept is taken from set theory. A singleton set has only one element.

Definition 1. A *singleton property* is a unique property instance representing a newly established relationship between two existing entities in one particular context.

For example, a new relationship is established for Bob Dylan and Sara Lownds according to two Wiki pages. We can consider each Wiki page as a context associated with the new relationship. We note that here we give examples of context and leave the questions of what exactly context is described and how to identify it for data publishers because those are subjective to them. As a result, we can create two singleton properties `isMarriedTo#1` and `isMarriedTo#2` to represent the new relationships associated with these two contexts. The statements asserting the new relationships become:

T_1: BobDylan isMarriedTo#1 SaraLownds, and
T_5: BobDylan isMarriedTo#2 SaraLownds.

Obviously, the number of such singleton properties would be as enormous as the number of facts and contexts in any real RDF datasets. We need to provide a mechanism to cluster them into groups for higher level abstraction. Such mechanism allows us to group similar singleton properties into a more general one.We observe that, although statements are fundamentally distinguishable based on their context, they do share common characteristics in their nature which are captured by generic properties. The relationship between singleton and generic properties can be considered from two different perspectives: the singleton property is either a sub property, or an instance of generic property.

Property instance. In this view, while generic properties are property whose extension contains a set of entity pairs, singleton properties are unique to one particular entity pair. Intuitively we can consider singleton properties as instances of generic properties. In that sense, a singleton property is interconnected to its generic property via `rdf:type`.

Sub property. Singleton property can be considered as a specialization, or sub property of a generic property in one particular context. In this case, if we create one singleton property for each fact via `rdfs:subPropertyOf`, the number of singleton property nodes below the generic property in the property hierarchy would become enormous. For example, YAGO has 23,770 facts [3] using the property `isMarriedTo`. A schema with such a large amount of child nodes in the property hierarchy is not desirable. This drawback does not happen with the property instance approach.

Therfore, we decide to stick with the property instance approach, where a singleton property and its generic property are interconnected via `rdf:type`. A generic property can be considered as a property class. The extension of generic property class is a set of singleton property instances created in all contexts. In the example described in Table 3, we use `rdf:type` in T_2 and T_6 of instead of `rdfs:subPropertyOf`.

T_2:	isMarriedTo#1	rdf:type	isMarriedTo, and
T_6:	isMarriedTo#2	rdf:type	isMarriedTo.

2.2 Asserting meta knowledge for triples

Here we consider various contextual dimensions for a statement, such as provenance, time, location, and certainty.

Provenance of a statement explains the origin of that statement [13, 18]. It includes many kinds of metadata for answering questions such as the ones listed in Table 1. For example, the triple T_1 and T_2 are extracted from the Wiki page of Bob Dylan and Sara Lownds. We can assert the provenance of two triples using property `hasSource` and `extractedOn` as follows.

T_3:	isMarriedTo#1	hasSource	wk:Bob_Dylan
T_4:	isMarriedTo#1	extractedOn	2009-06-07
T_7:	isMarriedTo#2	hasSource	wk:Sara_Dylan
T_8:	isMarriedTo#2	extractedOn	2009-08-08

Time. The validity of a statement may be associated with a specific time, or a time span. For example, a person is born at one specific time, and a marriage

[3] http://www.mpi-inf.mpg.de/yago-naga/yago/statistics.html

between two persons may last for one period of time. Here we represent the time span of the marriage between Bob Dylan and Sara Lownds using `hasStart` and `hasEnd`:

isMarriedTo#1	hasStart	1965-11-22 .
isMarriedTo#1	hasEnd	1977-06-29 .

Location. A statement may be associated with a spatial dimension. For example, the Wiki page of Sara Lownds stated that the marriage of Bob Dylan and Sara Lownds took place at Mineola, Long Island. We assert this meta knowledge for the singleton property `isMarriedTo#2` as follows.

isMarriedTo#2 tookPlaceAt Mineola .

Certainty. The certainty of a statement reflects the confidence of the authors while creating that statement. For example, if the confidence score of the tool extracting the statement T_2 is 0.7, we can represent it as follows.

isMarriedTo#2 hasScore 0.7 .

From the assertions created for provenance, time, location and certainty above, we observe that they share the same triple pattern, which is *singleton-property meta-property meta-value*. In our example, meta properties are `hasStart`, `hasEnd`, `hasSource`, `hasScore`, `tookPlaceAt`. We can generalize this pattern for representing all dimensions of meta knowledge as follows. **Singleton Graph Structure.** In general, given a fact (s, p, o), let `p#i` be the singleton property representing this fact in one particular context, `mp#j` be the meta property, `mv#j` be the value of meta property, the set of triples forming singleton graph structure asserting meta knowledge for this fact is provided in Table 4. We will use this singleton graph structure for querying meta knowledge in Section 4.

Table 4. Singleton graph structure asserting meta knowledge for data triple (s,p,o)

Triple type	**Subject**	**Predicate**	**Object**
Instantiating triple	p#i	rdf:type	p
Singleton triple	s	$p\#i$	o
Meta triple	p#i	$mp\#j$	$mv\#j$

2.3 Enforcing the singleton-ness of property instances

If the singleton property `isMarriedTo#1` is asserted in another triple such as `BarackObama isMarriedTo#1 MichelleObama`, this together with the existing assertion `isMarriedTo#1 hasStart 1965-11-22` would imply the marriage date of the Obamas is 1965-11-22, which is not true. In order to avoid this, we need to ensure the singleton property `isMarriedTo#1` occurs in only one triple.

This constraint has to be enforced for all URIs of singleton property instances. Data publishers may combine their URI prefix, the generic property name and the timestamp when the instance is created into the URI of a singleton property to make it unique. However, there are still cases that two instances may share the same URIs. Therefore, data publishers may employ the Universally Unique Identifier (UUIDs) [9] to ensure the singleton-ness of their property instances. The validation of this uniqueness constraint is straightforward, by counting the number of triple occurrences per singleton property. As the current RDF syntax does not allow blank nodes as properties, we do not represent singleton properties as blank nodes, although one advantage of using blank nodes in the property is decidability for reasoning [11].

3 Model-Theoretic Semantics

This section explains how the singleton property can fit to the existing formal semantics. We reuse the model-theoretic semantics described in [6] with three levels of interpretation: simple, RDF and RDFS. For each interpretation we add additional criteria for supporting singleton property. While we explain the new elements in detail, elements without further explanation remain as they are in the original model-theoretic semantics described in [6].

Given a vocabulary V, the original simple interpretation $\mathcal{I}$ consists of

- IR, a non-empty set of *resources*, alternatively called domain or universe of discourse of $\mathcal{I}$,
- IP, the set of *generic properties* of $\mathcal{I}$,
- I_{EXT}, a function assigning to each property a set of pairs from IR $I_{EXT} : IP \rightarrow 2^{IR \times IR}$ where $I_{EXT}(p)$ is called the *extension* of property p,
- I_S, a function, mapping URIs from V into the union set of IR and IP,
- I_L, a function from the typed literals from V into the set of resources IR,
- LV, a subset of IR, called the set of *literal values*.

We define IPs as a set of singleton properties and $I_{S\_EXT}(p_s)$ as the function mapping a singleton property into a pair of resources.

Simple interpretation of vocabulary V is an original simple interpretation of the vocabulary $V \cup V_{SIM}$ that satisfies the additional criteria:

- IPs, a subset of IR, called the set of *singleton properties* of $\mathcal{I}$,
- $I_{S\_EXT}(p_s)$, the extension of singleton property p_s, which is a singleton set, $I_{S\_EXT} : IPs \rightarrow IR \times IR$ and $|I_{S\_EXT}(p_s)| = 1$.

Note that the mapping function I_{S_EXT} is not a one-to-one mapping; multiple singleton properties may be mapped to the same pair of entities.

RDF interpretation of a vocabulary V is a simple interpretation of the vocabulary $V \cup V_{RDF}$ that satisfies the criteria from the original RDF interpretation and the following criteria:

- $x_s \in IPs$ if $\langle x_s, rdf\!:Property^{\mathcal{I}} \rangle \in I_{EXT}(rdf\!:type^{\mathcal{I}})$, and $|I_{S\_EXT}(x_s)| = 1$, this causes $IPs \subset IR$ because instances of class Property are resources.
- $x_s \in IPs$ if $\langle x_s, x^{\mathcal{I}} \rangle \in I_{EXT}(rdf\!:type^{\mathcal{I}})$, $x \in IP$, and $|I_{S\_EXT}(x_s)| = 1$, a singleton property x_s is an instance of a property x if they are interconnected by the property `rdf:type`, x is called a *generic property*.

A generic property may be singleton if it is connected to `rdf:Property` via `rdf:type` and it occurs in only one triple. However, a singleton property in general cannot be a generic property because a generic property is usually mapped to a non-singleton set of pairs, while the singleton property is mapped to one particular pair of entities, which is not equivalent to a pair (i.e., one element is not equivalent to a set of elements). Therefore, the set of singleton properties may contain some generic properties, but not vice versa.

Given the set of triples with singleton properties and their temporal assertions in Table 5, let V_{EX} be the vocabulary consisting of all the names of subjects, predicates and objects in those triples, the RDF interpretation of the vocabulary V_{EX} is provided in Table 6.

Table 5. Singleton property approach representing facts and their temporal assertions

Triple N_o	**Subject**	**Predicate**	**Object**
T_0	BobDylan	isMarriedTo	SaraLownds
T_1	BobDylan	isMarriedTo#1	SaraLownds
T_2	isMarriedTo#1	rdf:type	isMarriedTo
T_3	isMarriedTo#1	hasStart	1965-11-22
T_4	isMarriedTo#1	hasEnd	1977-06-29
T_5	BobDylan	isMarriedTo	CarolynDennis
T_6	BobDylan	isMarriedTo#2	CarolynDennis
T_7	isMarriedTo#2	rdf:type	isMarriedTo
T_8	isMarriedTo#2	hasStart	1986-06-##
T_9	isMarriedTo#2	hasEnd	1992-10-##

Table 6. RDF interpretation for the vocabulary V_{EX} from Table 5

IR	=	$\{\alpha, \beta, \gamma, \delta, \theta, \lambda\}$	I_S =	BobDylan	$\mapsto \alpha$
IP	=	$\{\delta, \theta, \lambda, \sigma, \phi\}$		SaraLownds	$\mapsto \beta$
LV	=	{1965-11-22, 1977-06-29,		CarolynDennis	$\mapsto \gamma$
		1986-06-##, 1992-10-##}		isMarriedTo	$\mapsto \delta$
I_{EXT}	=	$\theta \mapsto \{\langle \alpha, \beta \rangle\}$		isMarriedTo#1	$\mapsto \theta$
		$\lambda \mapsto \{\langle \alpha, \gamma \rangle\}$		isMarriedTo#2	$\mapsto \lambda$
		$\sigma \mapsto \{\langle \theta$, 1965-11-22 $\rangle$,		hasStart	$\mapsto \sigma$
		$\langle \lambda$, 1986-06-## $\rangle\}$		hasEnd	$\mapsto \phi$
		$\phi \mapsto \{\langle \theta$, 1977-06-29$\rangle$,			
		$\langle \lambda$, 1992-10-## $\rangle\}$			
		rdf:type $\mapsto \{\langle \theta, \delta \rangle, \langle \lambda, \delta \rangle\}$			
		$\delta \mapsto \{\langle \alpha, \beta \rangle, \langle \alpha, \gamma \rangle\}$			
IPs	=	$\{\theta, \lambda\}$			
I_{S_EXT}	=	$\theta \mapsto \langle \alpha, \beta \rangle$			
		$\lambda \mapsto \langle \alpha, \gamma \rangle$			

RDFS interpretation of a vocabulary V is an RDF interpretation of the vocabulary V $\cup$ V_{RDFS} that satisfies criteria from the original RDFS interpretation and the following criteria:

- If $x_s \in IPs$, $x \in IP$, and $\langle x_s, x\rangle \in I_{EXT}(rdf\!:type^{\mathcal{I}})$ then $I_{S\_EXT}(x_s) \in I_{EXT}(x)$. $I_{EXT}(x)$ is also called *class extension* of the generic property x. Class extension of a property is a set of singleton properties connected to that property via `rdf:type`.
- If $\langle x,y\rangle \in I_{EXT}(rdfs\!:domain^{\mathcal{I}})$, $\langle u,v\rangle \in I_{S\_EXT}(x_s)$, and $\langle x_s,x\rangle \in I_{EXT}(rdf\!:type^{\mathcal{I}})$, then $u \in I_{CEXT}(y)$ with $I_{CEXT}(y)$ is class extension of y.
  If u and v are interconnected by a singleton property x_s of generic property x, then u has the type of y, which is the domain of x.
- If $\langle x,y\rangle \in I_{EXT}(rdfs\!:range^{\mathcal{I}})$, $\langle u,v\rangle \in I_{S\_EXT}(x_s)$, and $\langle x_s,x\rangle \in I_{EXT}(rdf\!:type^{\mathcal{I}})$, then $v \in I_{CEXT}(y)$.
  If u and v are interconnected by a singleton property x_s of generic property x, then v has the type of y, which is the range of x.
- $I_{S\_EXT}(rdfs\!:subPropertyOf^{\mathcal{I}})$ is reflexive and transitive in *IPs*.
- If $\langle x,y\rangle \in I_{S\_EXT}(rdfs\!:subPropertyOf^{\mathcal{I}})$ then $x \in IPs, y \in IPs$, and $I_{S\_EXT}(x) = I_{S\_EXT}(y)$.
  If two singleton properties are interconnected by `rdfs:subPropertyOf`, then they must be mapped to the same pair of entities.

Classes such as `Resource`, `Literal`, and properties such as `subClassOf` are not discussed here because they are irrelevant to singleton properties.

Deduction rule. In the previous section, we instantiated singleton properties from its generic property via `rdf:type` and created meta triples for those singleton properties.

BobDylan	isMarriedTo#1	SaraLownds (*singleton triple*)
isMarriedTo#1	rdf:type	isMarriedTo (*singleton instantiation*)
BobDylan	isMarriedTo	SaraLownds (*original triple*)

Here we show that the first two triples can also yield the original triple. Formally, if $\langle x,y\rangle \in I_{EXT}(rdf\!:type^{\mathcal{I}})$ and $\langle u,v\rangle = I_{S\_EXT}(x)$ then $\langle u,v\rangle \in I_{EXT}(y)$. From the RDF interpretation provided in Table 6, we have:

$I_{S\_EXT}(isMarriedTo\#1) \in I_{EXT}(isMarriedTo)$ because

$I_{S\_EXT}(isMarriedTo\#1) = \langle\alpha,\beta\rangle$ and $I_{EXT}(isMarriedTo) = \{\langle\alpha,\beta\rangle,\langle\alpha,\gamma\rangle\}$ and $\langle\alpha,\beta\rangle \in \{\langle\alpha,\beta\rangle,\langle\alpha,\gamma\rangle\}$.

4 Querying

In Section 2, we described the singleton graph structure for representing meta knowledge. This section explains the principle for querying such meta knowledge based on that singleton graph structure. We will use the example from Table 5 for demonstrating how we query meta knowledge.

Since all these triples in singleton graph structure are represented in RDF, they can be queried using any RDF query language. Here we consider SPARQL as it is recommended for querying RDF by W3C [4].

In principle we can distinguish two basic types of query patterns: data vs. metadata. Both query patterns can be constructed as graph pattern in the SPARQL queries.

Data query contains graph patterns created from a set of factual data triples in the form of (s, p, o) by replacing any of subject s, property p or object o with variables. For example, given the set of data triples T_0 and T_5 of Table 5,

T_0:	BobDylan	isMarriedTo	SaraLownds
T_5:	BobDylan	isMarriedTo	CarolynDennis

we can construct a data query asking for spouses of BobDylan as follows.

```
SELECT ?obj
WHERE { BobDylan isMarriedTo ?obj}
```

This query type is commonly used in Semantic Web applications. Using the singleton property approach for representing meta knowledge of data triple, we can also easily query such meta knowledge.

Metadata query contains graph patterns created from a set of triples in the singleton graph structure by replacing any subject, property and object of any triple with variables. One sample of meta query pattern asking for the meta values of any meta property `mp` could be:

```
SELECT ?mp ?mv
WHERE {?pi rdf:type p . s ?pi o . ?pi ?mp ?mv . }
```

The query instance asking for the dates of BobDylan's marriages is as follows.

```
SELECT ?startOrEnd ?dates
WHERE {?pi rdf:type isMarriedTo .
BobDylan ?pi ?o . ?pi ?startOrEnd ?dates . }
```

In practice, one may mix data patterns and metadata patterns into one query pattern for more complicated queries. Next section will provide more queries of different kinds of metadata in detail.

5 Using Singleton Property in Existing Knowlege Bases

5.1 BKR and Provenance

The Biomedical Knowledge Repository (BKR) is an extensive knowledge base that integrates biomedical knowledge from multiple sources while tracking their provenance using a unified provenance framework [15, 16]. A triple in the BKR may be extracted from PubMed articles and is associated with a confidence score from its extraction tool. Given a triple (s, p, o) extracted from PMID#1 with confidence score 0.3, from PMID#2 with confidence score 0.8, the current representation of provenance of a triple with PaCE [16] is provided in Table 7, note that the confidence score cannot be represented by this approach.

The basic idea of PaCE is to create one instance of subject and object per context, and asserting the source of those instances. The source of the triple is inferred from the source of instances. This was proved to be better than the reification approach in terms of number of triples and query performance. However, this approach is limited in supporting different dimensions of meta

Table 7. PaCE approach for (s, p, o) with meta knowledge (PMID#1, 0.3) and (PMID#2, 0.8)

Subject	**Property**	**Object**	**Subject**	**Property**	**Object**
s_PMID#1	rdf:type	s	s_PMID#2	rdf:type	s
o_PMID#1	rdf:type	o	o_PMID#2	rdf:type	o
s_PMID#1	p	o_PMID#1	s_PMID#2	p	o_PMID#2
s_PMID#1	derivedFrom	PMID#1	s_PMID#2	derivedFrom	PMID#2
o_PMID#1	derivedFrom	PMID#1	o_PMID#2	derivedFrom	PMID#2

Table 8. Singleton Property approach for (s, p, o) with meta knowledge (PMID#1, 0.3) and (PMID#2, 0.8)

Subject	**Property**	**Object**	**Subject**	**Property**	**Object**
p#1	rdf:type	p	p#2	rdf:type	p
s	p#1	o	s	p#2	o
p#1	derivedFrom	PMID#1	p#2	derivedFrom	PMID#2
p#1	hasScore	0.3	p#2	hasScore	0.8

knowledge because it can only represent the source of a triple. Here we have at least two metadata associated with a triple, but it can only represent the source.

Using the Singleton Property approach, we can represent the complete metadata information as illustrated in Table 8. Moreover, if we need to represent more meta knowledge dimensions for the triple (s, p, o), we can simply add assertions into the singleton properties `p#1` and `p#2`.

Provenance query. Since the BKR integrates data from multiple sources, it is common to ask about the provenance of a triple, such as the sources, the publication date, the confidence score, and etc. For example, one may query the sources of a triple that has high confidence score (above 0.7). This query cannot be supported by PaCE approach because the confidence score is not present. Using the Singleton Property approach, and adopting the metadata query discussed in Section 4, we can create a query like this:

```
SELECT ?source ?score
WHERE {s p o . ?pi rdf:type p . ?pi derivedFrom ?source .
?pi hasScore ?score . FILTER (?score > 0.7) }
```

5.2 YAGO2 and Temporal-Spatial Enhancement

While YAGO [20] provides an extensive collection of factual triples extracted from Wiki and other sources, YAGO2 [8] enhances this knowledge base with temporal and spatial information for those factual triples. This knowledge base becomes aware of time and places and hence, is capable of answering more complex queries involving such meta knowledge.

Here we reuse the example from [8] to demonstrate the requirements of representing meta knowledge in YAGO2. We put the set of facts from the example into Table 9. YAGO2 uses fact identifiers to represent the facts, and asserts the occuring time and place of a fact by using its fact identifier as subjects of the meta assertions. It also provides a SPARQL-like query language which allows it to incorporate fact identifier in the query pattern. Here we propose to replace fact identifier by the singleton property in representing a statement and asserting its temporal and spatial information. This would enable the interoperability between this dataset with other RDF datasets and allow them to be queried using standard query language.

Table 9. YAGO2 uses fact id for representing fact and asserting meta knowledge

Fact Id	**Subject**	**Predicate**	**Object**
`#1`	GratefulDead	performed	TheClosingOfWinterland
`#2`	`#1`	occursIn	SanFrancisco
`#3`	`#1`	occursOnDate	1978-12-31

Table 10. Singleton property replaces fact id in asserting meta knowledge

Subject	**Predicate**	**Object**
`performed#1`	rdf:type	performed
GratefulDead	`performed#1`	TheClosingOfWinterland
`performed#1`	occursIn	SanFrancisco
`performed#1`	occursOnDate	1978-12-31

Temporal-spatial query in the YAGO can be specified in its query language SPARQL-like. For example, for finding the concerts that took place near San Francisco, one may create a SPARQL-like query as displayed on the left. We may also create an equivalent SPARQL query as displayed on the right for the triples using singleton property approach.

SPARQL-like pattern	**SPARQL pattern**
`?id: ?s performed ?o .`	`?performed#1` rdf:type performed .
`?id` occursIn ?l .	?s `?performed#1` ?o .
?l hasGeoCoordinates ?g .	`?performed#1` occursIn ?l .
SanFrancisco hasGeoCoordinates ?sf .	?l hasGeoCoordinates ?g .
?l near ?sf .	SanFrancisco hasGeoCoordinates ?sf .
	?l near ?sf .

Given that `near` is a proximate predicate, it may need to be elaborated in the graph pattern of SPARQL query on the right.

Here our purpose is to provide an RDF representation for meta triples in YAGO2 to make it compatible with existing RDF datasets and interoperable to other Semantic Web applications that use existing standards such as SPARQL. We do not attempt to compare the query performance or expressiveness between SPARQL and SPARQL-like language used in YAGO2.

6 Related Work

Many approaches have been proposed to address the problem of representing and querying meta knowledge of triples. We can divide these approaches into three main categories: triples [15, 16], quadruples [1, 3, 19] and quintuples [17] based on the number of elements in the structure each approach employs. Each approach reflects one perspective on how meta knowledge for triples could add elements into tuples. We will discuss about the contribution of each approach, and how our approach fits into the scheme.

Triples. Representing different dimensions of meta knowledge for triples using RDF triples in order to retain the compatibility and interoperability with existing Semantic Web knowledge bases, tools, languages and methods is the main goal of this kind of approach. The reification approach [12, 5] allows meta knowledge to be asserted from reified statements. The singleton property approach differs from the reification in that it provides a formal semantics. Moreover, it requires two triples for creating a singleton property and asserting the singleton tripe while a reified statement requires four triples. That would make it more efficient in terms of less number of triples, shorter query patterns, and hence, less number of joins in query processing.

Sahoo [16] proposes the PaCE approach for representing the provenance of a triple by creating different instances for its subject and object for different contexts and asserting provenance for those instances. The source of the triple is derived from the source of its individual components. The singleton property approach is similar to PaCE in that it creates different instances for capturing different contexts. However, the main difference between the two approaches has to do with which instances to create for each context. We ground our approach on shifting the focus from entities to properties. That is, within a new context, a new relationship is established between two already existing entities.

Quadruples. In the reification approach, we need to create statement instances and indicate subject, property, and object for those instances. Intuitively this verbosity can be avoided by adding one more element into a triple to make it quadruple. Named graph [1] and other work on top of named graph such as [3, 14] follow the approach, using the forth component to represent the provenance of a set of triples. Although technically we can restrict a named graph to a single triple and use it to assert meta knowledge to that triple, it does not naturally serve this purpose because originally the named graph is designed for representing provenance and trust of a set of triples. The number of named graphs would increase significantly (up to the number of triples in the dataset) and become dispropotionate with respect to its main purposes, managing provenance and signing trust certificates. On the other hand, the singleton property approach is complementary to named graph approach in representing provenance for different granularity levels of triples, one is for individual triple, the other one is for a set of triples. Data publishers may choose the approach that fulfills specific requirements of their applications.

Straccia [19] also annotates the meta knowledge such as temporal and certainty for RDF triples. We classify this approach into quadruples because it annotates every RDF triple with an annotation term. It proposes a new algebraic structures with well-defined operators for manipulating meta information. This approach is followed up with the RDFS reasoning supported by [2]. Our approach differs from this approach in that we leverage RDF triple for the representation of meta knowledge assertion, allowing them to be queried and entailed using existing languages and tools while this approach does not.

Quintuples. The RDF^+ approach [17] defines the abstract syntax of RDF^+ statement as a quintuple by extending the named graph quad with a statement identifier. The statement identifier is used as subject of the meta knowledge assertion, which is an RDF triple. Since the formal semantics is defined in RDF^+, mappings from RDF to RDF^+ and vice versa have to be made. Additionally, the SPARQL syntax and semantics have to be extended to support querying RDF^+. The singleton property approach differs from the RDF^+ approach in two main design points. Firstly, while a statement identifer is defined in the RDF^+ statement which is a quintuple, our approach represents singleton property instances in RDF triples. Because of that, our approach does not need any mapping as the RDF^+ does. Secondly, our approach does not require any extension to the syntax or semantics of SPARQL because it is completely compatible with SPARQL.

7 Conclusion

We have presented our approach for representing and querying meta knowledge using the singleton property. Regular RDF properties are viewed as generic properties in our approach, and the set of singleton properties are viewed as instances of those generic properties. Both singleton properties and meta knowledge assertions are represented using RDF syntax, allowing them to be compatible with existing RDF knowledge bases. The meaning of such a singleton property is defined in the formal semantics that is extended from the current model-theoretic semantics in all three steps of interpretation: simple, RDF, and RDFS. This singleton property approach also fits nicely the syntax of SPARQL query language. Because of those, we are able to demonstrate how this approach can be easily used for representing and query meta triples in two existing knowledge bases.

Therefore, adopting this approach in representing, querying, and sharing knowledge bases that are aware of meta knowledge would allow those knowledge bases to be compatible with a wide range of Semantic Web languages, tools, and methods. Please note that meta knowledge properties used in the paper are only for demonstration. When adopting this approach, one may want to use vocabularies of meta knowledge recommended by W3C such as PROV [10] or OWLTime [7] for enhancing the interoperability with other knowledge bases and applications.

Acknowledgement. This research was supported by the Intramural Research Program at NIH. We thank Jagannathan Srinivasan, Jayanta Banerjee, Pascal Hitzler, Chris Welty, Kemafor Anyanwu, Satya Sahoo, Ramakanth Kavuluru, Adila Alfa Krisnadhi, Sarasi Lalithena, Sujan Perera, Wenbo Wang, Lu

Chen, Pavan Kapanipathi, and Franklin Jose for fruitful discussion and feedback.

References

1. J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web*, pages 613–622. ACM, 2005.
2. C. V. Damásio and F. Ferreira. Practical rdf schema reasoning with annotated semantic web data. In *The Semantic Web–ISWC 2011*, pages 746–761. Springer, 2011.
3. G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, and V. Christophides. Coloring rdf triples to capture provenance. In *The Semantic Web-ISWC 2009*, pages 196–212. Springer, 2009.
4. S. Harris and A. Seaborne. Sparql 1.1 query language. *W3C Working Draft*, 2010.
5. P. Hayes and B. McBride. Rdf semantics, 2004.
6. P. Hitzler, M. Krotzsch, and S. Rudolph. *Foundations of semantic web technologies*. Chapman and Hall/CRC, 2011.
7. J. R. Hobbs and F. Pan. Time ontology in owl. *W3C working draft*, 27, 2006.
8. J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: a spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 2012.
9. P. J. Leach, M. Mealling, and R. Salz. A universally unique identifier (uuid) urn namespace. 2005.
10. T. Lebo, S. Sahoo, and D. McGuinness. Prov-o: The prov ontology. w3c working draft 03 may 2012. *W3C. http://www. w3. org/TR/prov-o*, 2012.
11. A. Mallea, M. Arenas, A. Hogan, and A. Polleres. On blank nodes. In *The Semantic Web–ISWC 2011*, pages 421–437. Springer, 2011.
12. F. Manola, E. Miller, and B. McBride. Rdf primer. *W3C recommendation*, 2004.
13. L. Moreau. The foundations for provenance on the web. *Foundations and Trends in Web Science*, 2(2–3):99–241, 2010.
14. P. Pediaditis, G. Flouris, I. Fundulaki, and V. Christophides. On explicit provenance management in rdf/s graphs. *TAPP*, 9:1–10, 2009.
15. S. Sahoo, V. Nguyen, O. Bodenreider, P. Parikh, T. Minning, and A. Sheth. A unified framework for managing provenance information in translational research. *BMC Bioinformatics*, 2011.
16. S. S. Sahoo, O. Bodenreider, P. Hitzler, A. Sheth, and K. Thirunarayan. Provenance context entity (pace): scalable provenance tracking for scientific rdf data. SSDBM'10, pages 461–470, 2010.
17. B. Schueler, S. Sizov, S. Staab, and D. T. Tran. Querying for meta knowledge. In *Proceedings of the 17th international conference on World Wide Web*, pages 625–634. ACM, 2008.
18. Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36, 2005.
19. U. Straccia, N. Lopes, G. Lukacsy, and A. Polleres. A general framework for representing and reasoning with annotated semantic web data. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), AAAI Press*, volume 28, 2010.
20. F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.